MADE: Masked Autoencoder for Distribution Estimation

Mathieu Germain   MATHIEU.GERMAIN2@USHERBROOKE.CA
Université de Sherbrooke, Canada

Karol Gregor   KAROL.GREGOR@GMAIL.COM
Google DeepMind

Iain Murray   I.MURRAY@ED.AC.UK
University of Edinburgh, United Kingdom

Hugo Larochelle   HUGO.LAROCHELLE@USHERBROOKE.CA
Université de Sherbrooke, Canada

Abstract

There has been a lot of recent interest in designing neural network models to estimate a distribution from a set of examples. We introduce a simple modification for autoencoder neural networks that yields powerful generative models. Our method masks the autoencoder's parameters to respect autoregressive constraints: each input is reconstructed only from previous inputs in a given ordering. Constrained this way, the autoencoder outputs can be interpreted as a set of conditional probabilities, and their product, the full joint probability. We can also train a single network that can decompose the joint probability in multiple different orderings. Our simple framework can be applied to multiple architectures, including deep ones. Vectorized implementations, such as on GPUs, are simple and fast. Experiments demonstrate that this approach is competitive with state-of-the-art tractable distribution estimators. At test time, the method is significantly faster and scales better than other autoregressive estimators.

1. Introduction

Distribution estimation is the task of estimating a joint distribution $p(\mathbf{x})$ from a set of examples $\{\mathbf{x}^{(t)}\}_{t=1}^{T}$, which is by definition a general problem. Many tasks in machine learning can be formulated as learning only specific properties of a joint distribution. Thus a good distribution estimator can be used in many scenarios, including classification (Schmah

et al., 2009), denoising or missing input imputation (Poon & Domingos, 2011; Dinh et al., 2014), data (e.g. speech) synthesis (Uria et al., 2015) and many others. The very nature of distribution estimation also makes it a particular challenge for machine learning. In essence, the curse of dimensionality has a distinct impact because, as the number of dimensions of the input space of $\mathbf{x}$ grows, the volume of space in which the model must provide a good answer for $p(\mathbf{x})$ exponentially increases.

Fortunately, recent research has made substantial progress on this task. Specifically, learning algorithms for a variety of neural network models have been proposed (Bengio & Bengio, 2000; Larochelle & Murray, 2011; Gregor & LeCun, 2011; Uria et al., 2013; 2014; Kingma & Welling, 2014; Rezende et al., 2014; Bengio et al., 2014; Gregor et al., 2014; Goodfellow et al., 2014; Dinh et al., 2014). These algorithms are showing great potential in scaling to high-dimensional distribution estimation problems. In this work, we focus our attention on *autoregressive* models (Section 3). Computing $p(\mathbf{x})$ exactly for a test example $\mathbf{x}$ is tractable with these models. However, the computational cost of this operation is still larger than typical neural network predictions for a D-dimensional input. For previous deep autoregressive models, evaluating $p(\mathbf{x})$ costs $O(D)$ times more than a simple neural network point predictor.

This paper's contribution is to describe and explore a simple way of adapting autoencoder neural networks that makes them competitive tractable distribution estimators that are faster than existing alternatives. We show how to mask the weighted connections of a standard autoencoder to convert it into a distribution estimator. The key is to use masks that are designed in such a way that the output is autoregressive for a given ordering of the inputs, i.e. that each input dimension is reconstructed solely from the dimensions preceding it in the

ordering. The resulting Masked Autoencoder Distribution Estimator (MADE) preserves the efficiency of a single pass through a regular autoencoder. Implementation on a GPU is straightforward, making the method scalable.

The single hidden layer version of MADE corresponds to the previously proposed autoregressive neural network of Bengio & Bengio (2000). Here, we go further by exploring deep variants of the model. We also explore training MADE to work simultaneously with multiple orderings of the input observations and hidden layer connectivity structures. We test these extensions across a range of binary datasets with hundreds of dimensions, and compare its statistical performance and scaling to comparable methods.

2. Autoencoders

A brief description of the basic autoencoder, on which this work builds upon, is required to clearly grasp what follows. In this paper, we assume that we are given a training set of examples $\{\mathbf{x}^{(t)}\}_{t=1}^{T}$. We concentrate on the case of binary observations, where for every D-dimensional input $\mathbf{x}$, each input dimension x_d belongs in $\{0, 1\}$. The motivation is to learn hidden representations of the inputs that reveal the statistical structure of the distribution that generated them.

An autoencoder attempts to learn a feed-forward, hidden representation $\mathbf{h}(\mathbf{x})$ of its input $\mathbf{x}$ such that, from it, we can obtain a reconstruction $\widehat{\mathbf{x}}$ which is as close as possible to $\mathbf{x}$. Specifically, we have

$$
\begin{aligned}
\mathbf{h}(\mathbf{x}) &= \mathbf{g}(\mathbf{b} + \mathbf{W}\mathbf{x}) & (1) \\
\widehat{\mathbf{x}} &= \mathrm{sigm}(\mathbf{c} + \mathbf{V}\mathbf{h}(\mathbf{x})), & (2)
\end{aligned}
$$

where $\mathbf{W}$ and $\mathbf{V}$ are matrices, $\mathbf{b}$ and $\mathbf{c}$ are vectors, $\mathbf{g}$ is a nonlinear activation function and $\mathrm{sigm}(a) = 1/(1 + \exp(-a))$. Thus, $\mathbf{W}$ represents the connections from the input to the hidden layer, and $\mathbf{V}$ represents the connections from the hidden to the output layer.

To train the autoencoder, we must first specify a training loss function. For binary observations, a natural choice is the cross-entropy loss:

$$
\ell(\mathbf{x}) = \sum_{d=1}^{D} -x_d \log \widehat{x}_d - (1-x_d) \log(1-\widehat{x}_d). \quad (3)
$$

By treating $\widehat{x}_d$ as the model's probability that x_d is 1, the cross-entropy can be understood as taking the form of a negative log-likelihood function. Training the autoencoder corresponds to optimizing the parameters $\{\mathbf{W}, \mathbf{V}, \mathbf{b}, \mathbf{c}\}$ to reduce the average loss on the training examples, usually with (mini-batch) stochastic gradient descent.

One advantage of the autoencoder paradigm is its flexibility. In particular, it is straightforward to obtain a deep autoencoder by inserting more hidden layers between the input

and output layers. Its main disadvantage is that the representation it learns can be trivial. For instance, if the hidden layer is at least as large as the input, hidden units can each learn to "copy" a single input dimension, so as to reconstruct all inputs perfectly at the output layer. One obvious consequence of this observation is that the loss function of Equation 3 isn't in fact a proper log-likelihood function. Indeed, since perfect reconstruction could be achieved, the implied data 'distribution' $q(\mathbf{x}) = \prod_d \widehat{x}_d^{x_d} (1-\widehat{x}_d)^{1-x_d}$ could be learned to be 1 for any $\mathbf{x}$ and thus not be properly normalized ($\sum_{\mathbf{x}} q(\mathbf{x}) \neq 1$).

3. Distribution Estimation as Autoregression

An interesting question is what property we could impose on the autoencoder, such that its output can be used to obtain valid probabilities. Specifically, we'd like to be able to write $p(\mathbf{x})$ in such a way that it could be computed based on the output of a properly corrected autoencoder.

First, we can use the fact that, for any distribution, the probability product rule implies that we can always decompose it into the product of its nested conditionals

$$
p(\mathbf{x}) = \prod_{d=1}^{D} p(x_d \,|\, \mathbf{x}_{<d}), \quad (4)
$$

where $\mathbf{x}_{<d} = [x_1, \ldots, x_{d-1}]^{\top}$.

By defining $p(x_d = 1 \,|\, \mathbf{x}_{<d}) = \hat{x}_d$, and thus $p(x_d = 0 \,|\, \mathbf{x}_{<d}) = 1 - \hat{x}_d$, the loss of Equation 3 becomes a valid negative log-likelihood:

$$
\begin{aligned}
-\log p(\mathbf{x}) &= \sum_{d=1}^{D} -\log p(x_d \,|\, \mathbf{x}_{<d}) \\
&= \sum_{d=1}^{D} -x_d \log p(x_d = 1 \,|\, \mathbf{x}_{<d}) \\
&\qquad - (1-x_d) \log p(x_d = 0 \,|\, \mathbf{x}_{<d}) \\
&= \ell(\mathbf{x}).
\end{aligned} \quad (5)
$$

This connection provides a way to define autoencoders that can be used for distribution estimation. Each output $\widehat{x}_d = p(x_d \,|\, \mathbf{x}_{<d})$ must be a function taking as input $\mathbf{x}_{<d}$ only and outputting the probability of observing value x_d at the dth dimension. In particular, the autoencoder forms a proper distribution if each output unit $\hat{x}_d$ only depends on the previous input units $\mathbf{x}_{<d}$, and not the other units $\mathbf{x}_{\geq d} = [x_d, \ldots, x_D]^{\top}$.

We refer to this property as the *autoregressive property*, because computing the negative log-likelihood (5) is equivalent to sequentially predicting (regressing) each dimension of input $\mathbf{x}$.

4. Masked Autoencoders

The question now is how to modify the autoencoder so as to satisfy the autoregressive property. Since output $\hat{x}_d$ must depend only on the preceding inputs $\mathbf{x}_{<d}$, it means that there must be no computational path between output unit $\hat{x}_d$ and any of the input units $x_d, \ldots, x_D$. In other words, for each of these paths, at least one connection (in matrix $\mathbf{W}$ or $\mathbf{V}$) must be 0.

A convenient way of zeroing connections is to elementwise-multiply each matrix by a binary mask matrix, whose entries that are set to 0 correspond to the connections we wish to remove. For a single hidden layer autoencoder, we write

$$
\begin{aligned}
\mathbf{h}(\mathbf{x}) &= \mathbf{g}(\mathbf{b} + (\mathbf{W} \odot \mathbf{M^W})\mathbf{x}) & (6) \\
\hat{\mathbf{x}} &= \mathrm{sigm}(\mathbf{c} + (\mathbf{V} \odot \mathbf{M^V})\mathbf{h}(\mathbf{x})) & (7)
\end{aligned}
$$

where $\mathbf{M^W}$ and $\mathbf{M^V}$ are the masks for $\mathbf{W}$ and $\mathbf{V}$ respectively. It is thus left to the masks $\mathbf{M^W}$ and $\mathbf{M^V}$ to satisfy the autoregressive property.

To impose the autoregressive property we first assign each unit in the hidden layer an integer m between 1 and $D-1$ inclusively. The kth hidden unit's number $m(k)$ gives the maximum number of input units to which it can be connected. We disallow $m(k) = D$ since this hidden unit would depend on all inputs and could not be used in modelling any of the conditionals $p(x_d \,|\, \mathbf{x}_{<d})$. Similarly, we exclude $m(k) = 0$, as it would create constant hidden units.

The constraints on the maximum number of inputs to each hidden unit are encoded in the matrix masking the connections between the input and hidden units:

$$
M^{\mathbf{W}}_{k,d} = 1_{m(k) \geq d} = \begin{cases} 1 & \text{if } m(k) \geq d \\ 0 & \text{otherwise,} \end{cases} \quad (8)
$$

for $d \in \{1, \ldots, D\}$ and $k \in \{1, \ldots, K\}$. Overall, we need to encode the constraint that the dth output unit is only connected to $\mathbf{x}_{<d}$ (and thus not to $\mathbf{x}_{\geq d}$). Therefore the output weights can only connect the dth output to hidden units with $m(k) < d$, i.e. units that are connected to at most $d-1$ input units. These constraints are encoded in the output mask matrix:

$$
M^{\mathbf{V}}_{d,k} = 1_{d > m(k)} = \begin{cases} 1 & \text{if } d > m(k) \\ 0 & \text{otherwise,} \end{cases} \quad (9)
$$

again for $d \in \{1, \ldots, D\}$ and $k \in \{1, \ldots, K\}$. Notice that, from this rule, no hidden units will be connected to the first output unit $\hat{x}_1$, as desired.

From these mask constructions, we can easily demonstrate that the corresponding masked autoencoder satisfies the autoregressive property. First, we note that, since the masks $\mathbf{M^V}$ and $\mathbf{M^W}$ represent the network's connectivity, their matrix product $\mathbf{M^{V,W}} = \mathbf{M^V}\mathbf{M^W}$ represents the connectivity between the input and the output layer. Specifically,

$M^{\mathbf{V,W}}_{d',d}$ is the number of network paths between output unit $\hat{x}_{d'}$ and input unit x_d. Thus, to demonstrate the autoregressive property, we need to show that $\mathbf{M^{V,W}}$ is strictly lower diagonal, i.e. $M^{\mathbf{V,W}}_{d',d}$ is 0 if $d' \leq d$. By definition of the matrix product, we have:

$$
M^{\mathbf{V,W}}_{d',d} = \sum_{k=1}^{K} M^{\mathbf{V}}_{d',k} M^{\mathbf{W}}_{k,d} = \sum_{k=1}^{K} 1_{d' > m(k)} 1_{m(k) \geq d}. \quad (10)
$$

If $d' \leq d$, then there are no values for $m(k)$ such that it is both strictly less than d' and greater or equal to d. Thus $M^{\mathbf{V,W}}_{d',d}$ is indeed 0.

Constructing the masks $\mathbf{M^V}$ and $\mathbf{M^W}$ only requires an assignment of the $m(k)$ values to each hidden unit. One could imagine trying to assign an (approximately) equal number of units to each legal value of $m(k)$. In our experiments, we instead set $m(k)$ by sampling from a uniform discrete distribution defined on integers from 1 to $D-1$, independently for each of the K hidden units.

Previous work on autoregressive neural networks have also found it advantageous to use direct connections between the input and output layers (Bengio & Bengio, 2000). In this context, the reconstruction becomes:

$$
\hat{\mathbf{x}} = \mathrm{sigm}(\mathbf{c} + (\mathbf{V} \odot \mathbf{M^V})\mathbf{h}(\mathbf{x}) + (\mathbf{A} \odot \mathbf{M^A})\mathbf{x}), \quad (11)
$$

where $\mathbf{A}$ is the parameter connection matrix and $\mathbf{M^A}$ is its mask matrix. To satisfy the autoregressive property, $\mathbf{M^A}$ simply needs to be a strictly lower diagonal matrix, filled otherwise with ones. We used such direct connections in our experiments as well.

4.1. Deep MADE

One advantage of the masked autoencoder framework described in the previous section is that it naturally generalizes to deep architectures. Indeed, as we'll see, by assigning a maximum number of connected inputs to all units across the deep network, masks can be similarly constructed so as to satisfy the autoregressive property.

For networks with $L > 1$ hidden layers, we use superscripts to index the layers. The first hidden layer matrix (previously $\mathbf{W}$) will be denoted $\mathbf{W}^1$, the second hidden layer matrix will be $\mathbf{W}^2$, and so on. The number of hidden units (previously K) in each hidden layer will be similarly indexed as K^l, where l is the hidden layer index. We will also generalize the notation for the maximum number of connected inputs of the kth unit in the lth layer to $m^l(k)$.

We've already discussed how to define the first layer's mask matrix such that it ensures that its kth unit is connected to at most $m(k)$ (now $m^1(k)$) inputs. To impose the same property on the second hidden layer, we must simply make sure that each unit k' is only connected to first layer units
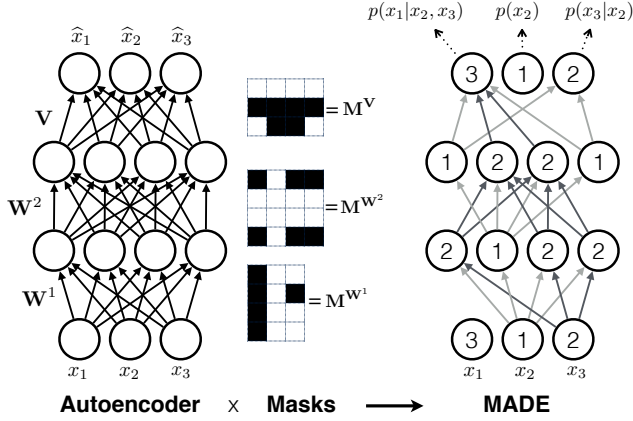
Figure 1. **Left: Conventional three hidden layer autoencoder**. Input in the bottom is passed through fully connected layers and point-wise nonlinearities. In the final top layer, a reconstruction specified as a probability distribution over inputs is produced. As this distribution depends on the input itself, a standard autoencoder cannot predict or sample new data. **Right: MADE**. The network has the same structure as the autoencoder, but a set of connections is removed such that each input unit is only predicted from the previous ones, using multiplicative binary masks ($\mathbf{M^{W^1}}, \mathbf{M^{W^2}}, \mathbf{M^V}$). In this example, the ordering of the input is changed from 1,2,3 to 3,1,2. This change is explained in section 4.2, but is not necessary for understanding the basic principle. The numbers in the hidden units indicate the maximum number of inputs on which the kth unit of layer l depends. The masks are constructed based on these numbers (see Equations 12 and 13). These masks ensure that MADE satisfies the autoregressive property, allowing it to form a probabilistic model, in this example $p(\mathbf{x}) = p(x_2)\, p(x_3|x_2)\, p(x_1|x_2, x_3)$. Connections in light gray correspond to paths that depend only on 1 input, while the dark gray connections depend on 2 inputs.

connected to at most $m^2(k')$ inputs, i.e. the first layer units such that $m^1(k) \leq m^2(k')$.

One can generalize this rule to any layer l, as follows:

$$M^{\mathbf{W}^l}_{k',k} = 1_{m^l(k') \geq m^{l-1}(k)} = \begin{cases} 1 & \text{if } m^l(k') \geq m^{l-1}(k) \\ 0 & \text{otherwise.} \end{cases}$$
(12)

Also, taking $l = 0$ to mean the input layer and defining $m^0(d) = d$ (which is intuitive, since the dth input unit indeed takes its values from the d first inputs), this definition also applies for the first hidden layer weights. As for the output mask, we simply need to adapt its definition by using the connectivity constraints of the last hidden layer $m^L(k)$ instead of the first:

$$M^{\mathbf{V}}_{d,k} = 1_{d > m^L(k)} = \begin{cases} 1 & \text{if } d > m^L(k) \\ 0 & \text{otherwise.} \end{cases}$$
(13)

Like for the single hidden layer case, the values for $m^l(k)$ for each hidden layer $l \in \{1, \ldots, L\}$ are sampled uniformly. To avoid unconnected units, the value for $m^l(k)$ is sampled

to be greater than or equal to the minimum connectivity at the previous layer, i.e. $\min_{k'} m^{l-1}(k')$.

4.2. Order-agnostic training

So far, we've assumed that the conditionals modelled by MADE were consistent with the natural ordering of the dimensions of $\mathbf{x}$. However, we might be interested in modelling the conditionals associated with an arbitrary ordering of the input's dimensions.

Specifically, Uria et al. (2014) have shown that training an autoregressive model on *all* orderings can be beneficial. We refer to this approach as order-agnostic training. It can be achieved by sampling an ordering before each stochastic/minibatch gradient update of the model. There are two advantages of this approach. Firstly, missing values in partially observed input vectors can be imputed efficiently: we invoke an ordering where observed dimensions are all before unobserved ones, making inference straightforward. Secondly, an ensemble of autoregressive models can be constructed on the fly, by exploiting the fact that the conditionals for two different orderings are not guaranteed to be exactly consistent (and thus technically correspond to slightly different models). An ensemble is then easily obtained by sampling a set of orderings, computing the probability of $\mathbf{x}$ under each ordering and averaging.

Conveniently, in MADE, the ordering is simply represented by the vector $\mathbf{m}^0 = [m^0(1), \ldots, m^0(D)]$. Specifically, $m^0(d)$ corresponds to the position of the original d^{th} dimension of $\mathbf{x}$ in the product of conditionals. Thus, a random ordering can be obtained by randomly permuting the ordered vector $[1, \ldots, D]$. From these values of each $\mathbf{m}^0$, the first hidden layer mask matrix can then be created. During order-agnostic training, randomly permuting the last value of $\mathbf{m}^0$ again is sufficient to obtain a new random ordering.

4.3. Connectivity-agnostic training

One advantage of order-agnostic training is that it effectively allows us to train as many models as there are orderings, using a common set of parameters. This can be exploited by creating ensembles of models at test time.

In MADE, in addition to choosing an ordering, we also have to choose each hidden unit's connectivity constraint $m^l(k)$. Thus, we could imaging training MADE to also be agnostic of the connectivity pattern generated by these constraints. To achieve this, instead of sampling the values of $m^l(k)$ for all units and layers once and for all before training, we actually resample them for each training example or minibatch. This is still practical, since the operation of creating the masks is easy to parallelize. Denoting $\mathbf{m}^l = [m^l(1), \ldots, m^l(K^l)]$, and assuming an element-wise and parallel implementation of the operation $1_{a \geq b}$ for vectors, such that $\mathbf{1_{a \geq b}}$ is a matrix

Algorithm 1 Computation of $p(\mathbf{x})$ and learning gradients for MADE with order and connectivity sampling. D is the size of the input, L the number of hidden layers and K the number of hidden units.

Input: training observation vector $\mathbf{x}$
Output: $p(\mathbf{x})$ and gradients of $-\log p(\mathbf{x})$ on parameters

\# Sampling $\mathbf{m}^l$ vectors
$\mathbf{m}^0 \leftarrow \text{shuffle}([1, \ldots, D])$
for l from 1 to L **do**
   for k from 1 to K^l **do**
      $m^l(k) \leftarrow \text{Uniform}([\min_{k'} m^{l-1}(k'), \ldots, D-1])$
   end for
end for

\# Constructing masks for each layer
for l from 1 to L **do**
   $\mathbf{M}^{\mathbf{W}^l} \leftarrow \mathbf{1}_{\mathbf{m}^l \geq \mathbf{m}^{l-1}}$
end for
$\mathbf{M}^{\mathbf{V}} \leftarrow \mathbf{1}_{\mathbf{m}^0 > \mathbf{m}^L}$

\# Computing $p(\mathbf{x})$
$\mathbf{h}^0(\mathbf{x}) \leftarrow \mathbf{x}$
for l from 1 to L **do**
   $\mathbf{h}^l(\mathbf{x}) \leftarrow \mathbf{g}(\mathbf{b}^l + (\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l})\mathbf{h}^{l-1}(\mathbf{x}))$
end for
$\widehat{\mathbf{x}} \leftarrow \text{sigm}(\mathbf{c} + (\mathbf{V} \odot \mathbf{M}^{\mathbf{V}})\mathbf{h}^L(\mathbf{x}))$
$p(\mathbf{x}) \leftarrow \exp\left(\sum_{d=1}^{D} x_d \log \widehat{x}_d + (1-x_d)\log(1-\widehat{x}_d)\right)$

\# Computing gradients of $-\log p(\mathbf{x})$
$\text{tmp} \leftarrow \widehat{\mathbf{x}} - \mathbf{x}$
$\delta\mathbf{c} \leftarrow \text{tmp}$
$\delta\mathbf{V} \leftarrow \left(\text{tmp } \mathbf{h}^L(\mathbf{x})^\top\right) \odot \mathbf{M}^{\mathbf{V}}$
$\text{tmp} \leftarrow (\text{tmp}^\top (\mathbf{V} \odot \mathbf{M}^{\mathbf{V}}))^\top$
for l from L to 1 **do**
   $\text{tmp} \leftarrow \text{tmp} \odot \mathbf{g}'(\mathbf{b}^l + (\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l})\mathbf{h}^{l-1}(\mathbf{x}))$
   $\delta\mathbf{b}^l \leftarrow \text{tmp}$
   $\delta\mathbf{W}^l \leftarrow \left(\text{tmp } \mathbf{h}^{l-1}(\mathbf{x})^\top\right) \odot \mathbf{M}^{\mathbf{W}^l}$
   $\text{tmp} \leftarrow (\text{tmp}^\top (\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l}))^\top$
end for
return $p(\mathbf{x}), \delta\mathbf{b}^1, \ldots, \delta\mathbf{b}^L, \delta\mathbf{W}^1, \ldots, \delta\mathbf{W}^L, \delta\mathbf{c}, \delta\mathbf{V}$

---

whose i, j element is $\mathbf{1}_{a_i \geq b_j}$, then the hidden layer masks are simply $\mathbf{M}^{\mathbf{W}^l} = \mathbf{1}_{\mathbf{m}^l \geq \mathbf{m}^{l-1}}$.

By resampling the connectivity of hidden units for every update, each hidden unit will have a constantly changing number of incoming inputs during training. However, the absence of a connection is indistinguishable from an instantiated connection to a zero-valued unit, which could confuse the neural network during training. In a similar situation, Uria et al. (2014) informed each hidden unit which units

were providing input with binary indicator variables, connected with additional learnable weights. We considered applying a similar strategy, using companion weight matrices $\mathbf{U}^l$, that are also masked by $\mathbf{M}^{\mathbf{W}^l}$ but connected to a constant one-valued vector:

$$\mathbf{h}^l(\mathbf{x}) = \mathbf{g}(\mathbf{b}^l + (\mathbf{W}^l \odot \mathbf{M}^{\mathbf{W}^l})\mathbf{h}^{l-1}(\mathbf{x}) + (\mathbf{U}^l \odot \mathbf{M}^{\mathbf{W}^l})\mathbf{1}) \tag{14}$$

An analogous parametrization of the output layer was also employed. These connectivity conditioning weights were only sometimes useful. In our experiments, we treated the choice of using them as a hyperparameter.

Moreover, we've found in our experiments that sampling masks for every example could sometimes over-regularize MADE and provoke underfitting. To fix this issue, we also considered sampling from only a finite list of masks. During training, MADE cycles through this list, using one for every update. At test time, we then average probabilities obtained for all masks in the list.

Algorithm 1 details how $p(\mathbf{x})$ is computed by MADE, as well as how to obtain the gradient of $\ell(\mathbf{x})$ for stochastic gradient descent training. For simplicity, the pseudocode assumes order-agnostic and connectivity-agnostic training, doesn't assume the use of conditioning weight matrices or of direct input/output connections. Figure 1 also illustrates an example of such a two-layer MADE network, along with its $m^l(k)$ values and its masks.

5. Related Work

There has been a lot of recent work on exploring the use of feed-forward, autoencoder-like neural networks as probabilistic generative models. Part of the motivation behind this research is to test the common assumption that the use of models with probabilistic latent variables and intractable partition functions (such as the restricted Boltzmann machine (Salakhutdinov & Murray, 2008)), is a necessary evil in designing powerful generative models for high-dimensional data.

The work on the neural autoregressive distribution estimator or NADE (Larochelle & Murray, 2011) has illustrated that feed-forward architectures can in fact be used to form state-of-the-art and even tractable distribution estimators.

Recently, a deep extension of NADE was proposed, improving even further the state-of-the-art in distribution estimation (Uria et al., 2014). This work introduced a randomized training procedure, which (like MADE) has nearly the same cost per iteration as a standard autoencoder. Unfortunately, deep NADE models still require D feed-forward passes through the network to evaluate the probability $p(\mathbf{x})$ of a D-dimensional test vector. The computation of the first hidden layer's activations can be shared across these passes,

Table 1. Complexity of the different models in Table 6, to compute an exact test negative log-likelihood. R is the number of orderings used, D is the input size, and K is the hidden layer size (assuming equally sized hidden layers).

Model	O_{NLL}
RBM 25 CD steps	$O(\min(2^D K, D2^K))$
DARN	$O(2^K D)$
NADE (fixed order)	$O(DK)$
EoNADE 1hl, R ord.	$O(RDK)$
EoNADE 2hl, R ord.	$O(RDK^2)$
MADE 1hl, 1 ord.	$O(DK+D^2)$
MADE 2hl, 1 ord.	$O(DK+K^2+D^2)$
MADE 1hl, R ord.	$O(R(DK+D^2))$
MADE 2hl, R ord.	$O(R(DK+K^2+D^2))$

Table 2. Number of input dimensions and numbers of examples in the train, validation, and test splits.

Name	# Inputs	Train	Valid.	Test
Adult	123	5000	1414	26147
Connect4	126	16000	4000	47557
DNA	180	1400	600	1186
Mushrooms	112	2000	500	5624
NIPS-0-12	500	400	100	1240
OCR-letters	128	32152	10000	10000
RCV1	150	40000	10000	150000
Web	300	14000	3188	32561

although is slower in practice than evaluating a single pass in a standard autoencoder. In deep networks with K hidden units per layer, it costs $O(DK^2)$ to evaluate a test vector.

Deep AutoRegressive Networks (DARN, Gregor et al., 2014), also provide probabilistic models with roughly the same training costs as standard autoencoders. DARN's latent representation consist of binary, stochastic hidden units. While simulating from these models is fast, evaluation of exact test probabilities requires summing over all configurations of the latent representation, which is exponential in computation. Monte Carlo approximation is thus recommended.

The main advantage of MADE is that evaluating probabilities retains the efficiency of autoencoders, with minor additional cost for simple masking operations. Table 1 lists the computational complexity for exact computation of probabilities for various models. DARN and RBMs are exponential in dimensionality of the hiddens or data, whereas NADE and MADE are polynomial. MADE only requires one pass through the autoencoder rather than the D passes required by NADE. In practice, we also observe that the single-layer MADE is an order of magnitude faster than a one-layer NADE, for the same hidden layer size, despite NADE sharing computation to get the same asymptotic scaling. NADE's computations cannot be vectorized as efficiently. The deep versions of MADE also have better scaling than NADE at test time. The training costs for MADE, DARN, and deep NADE will all be similar.

Before the work on NADE, Bengio & Bengio (2000) proposed a neural network architecture that corresponds to the special case of a single hidden layer MADE model, without randomization of input ordering and connectivity. A contribution of our work is to go beyond this special case, exploring deep variants and order/connectivity-agnostic training.

An interesting interpretation of the autoregressive mask sampling is as a structured form of dropout regularization (Srivastava et al., 2014). Specifically, it bears similarity with the masking in dropconnect networks (Wan et al., 2013). The exception is that the masks generated here must guaranty the autoregressive property of the autoencoder, while in Wan et al. (2013), each element in the mask is generated independently.

6. Experiments

To test the performance of our model we considered two different benchmarks: a suite of UCI binary datasets, and the binarized MNIST dataset. The code to reproduce the experiments of this paper is available at https://github.com/mgermain/MADE/releases/tag/ICML2015. The results reported here are the average negative log-likelihood on the test set of each respective dataset. All experiments were made using stochastic gradient descent (SGD) with mini-batches of size 100 and a lookahead of 30 for early stopping.

6.1. UCI evaluation suite

We use the binary UCI evaluation suite that was first put together in Larochelle & Murray (2011). It's a collection of 7 relatively small datasets from the University of California, Irvine machine learning repository and the OCR-letters dataset from the Stanford AI Lab. Table 2 gives an overview of the scale of those datasets and the way they were split.

The experiments were run with networks of 500 units per hidden layer, using the adadelta learning update (Zeiler, 2012) with a decay of 0.95. The other hyperparameters were varied as Table 3 indicates. We note as *# of masks* the number of different masks through which MADE cycles during training. In the no limit case, masks are sampled on the fly and never explicitly reused unless re-sampled by chance. In this situation, at validation and test time, 300 and 1000 sampled masks are used for averaging probabilities.

Table 3. UCI Grid Search

Hyperparameter	Values tried
# Hidden Layer	1, 2
Activation function	ReLU, Softplus
Adadelta epsilon	$10^{-5}, 10^{-7}, 10^{-9}$
Conditioning Weights	True, False
# of orderings	1, 8, 16, 32, No Limit

Table 5. Binarized MNIST Grid Search

Hyperparameter	Values tried
# Hidden Layer	1, 2
Learning Rate	0.1, 0.05, 0.01, 0.005
# of masks	1, 2, 4, 8, 16, 32, 64

The results are reported in Table 4. We see that MADE is among the best performing models on half of the datasets and is competitive otherwise. To reduce clutter, we have not reported standard deviations, which were fairly small and consistent across models. However, for completeness we report standard deviations in a separate table in the supplementary materials.

An analysis of the hyperparameters selected for each dataset reveals no clear winner. However, we do see from Table 4 that when the mask sampling helps, it helps quite a bit and when it does not, the impact is negligible on all but OCR-letters. Another interesting note is that the conditioning weights had almost no influence except on NIPS-0-12 where it helped.

6.2. Binarized MNIST evaluation

The version of MNIST we used is the one binarized by Salakhutdinov & Murray (2008). MNIST is a set of 70,000 hand written digits of 28×28 pixels. We use the same split as in Larochelle & Murray (2011), consisting of 50,000 for the training set, 10,000 for the validation set and 10,000 for the test set.

Experiments were run using the adagrad learning update (Duchi et al., 2010), with an epsilon of 10^{-6}. Since MADE is much more efficient than NADE, we considered varying the hidden layer size from 500 to 8000 units. Seeing that increasing the number of units tended to always help, we used 8000. Even with such a large hidden layer, our GPU implementation of MADE was quite efficient. Using a single mask, one training epoch requires about 14 and 44 seconds, for one hidden layer and two hidden layer MADE respectively. Using 32 sampled masks, training time increases to 33 and 100 respectively. These timings are all less than our GPU implementation of the 500 hidden units NADE model, which requires about 130 seconds per epoch. These timings were obtained on a K20 NVIDIA GPU.

Building on what we learned on the UCI experiments, we set the activation function to be ReLU and the conditioning weights were not used. The hyperparameters that were varied are in Table 5.

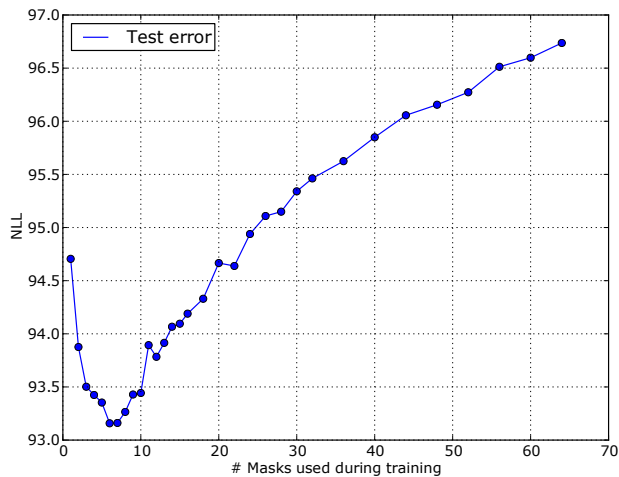The results are reported in Table 6, alongside other results

Figure 2. Impact of the number of masks used with a single hidden layer, 500 hidden units network, on binarized MNIST.

taken from the literature. Again, despite its tractability, MADE is competitive with other models. Of note is the fact that the best MADE model outperforms the single layer NADE network, which was otherwise the best model among those requiring only a single feed-forward pass to compute log probabilities.

In these experiments, we clearly observed the over-regularization phenomenon from using too many masks. When more than four orderings were used, the deeper variant of MADE always yielded better results. For the two layer model, adding masks during training helped up to 64, at which point the negative log-likelihood started to increase. We observed a similar pattern for the single layer model, but in this case the dip was around 8 masks. Figure 2 illustrates this behaviour more precisely for a single layer MADE with 500 hidden units, trained by only varying the number of masks used and the size of the mini-batches (83, 100, 128).

We randomly sampled 100 digits from our best performing model from Table 6 and compared them with their nearest neighbor in the training set (Figure 3), to ensure that the generated samples are not simple memorization. Each row of digits uses a different mask that was seen at training time by the network.

Table 4. Negative log-likelihood test results of different models on multiple datasets. The best result as well as any other result with an overlapping confidence interval is shown in bold. Note that since the variance of DARN was not available, we considered it to be zero.

| Model	Adult	Connect4	DNA	Mushrooms	NIPS-0-12	OCR-letters	RCV1	Web
MoBernoullis	20.44	23.41	98.19	14.46	290.02	40.56	47.59	30.16
RBM	16.26	22.66	96.74	15.15	277.37	43.05	48.88	29.38
FVSBN	**13.17**	12.39	83.64	10.27	276.88	39.30	49.84	29.35
NADE (fixed order)	**13.19**	11.99	84.81	9.81	**273.08**	**27.22**	46.66	28.39
EoNADE 1hl (16 ord.)	**13.19**	12.58	82.31	9.69	**272.39**	**27.32**	**46.12**	**27.87**
DARN	13.19	11.91	81.04	**9.55**	274.68	≈28.17	≈**46.10**	≈28.83
MADE	**13.12**	**11.90**	83.63	9.68	280.25	28.34	47.10	28.53
MADE mask sampling	**13.13**	**11.90**	**79.66**	9.69	277.28	30.04	46.74	**28.25**
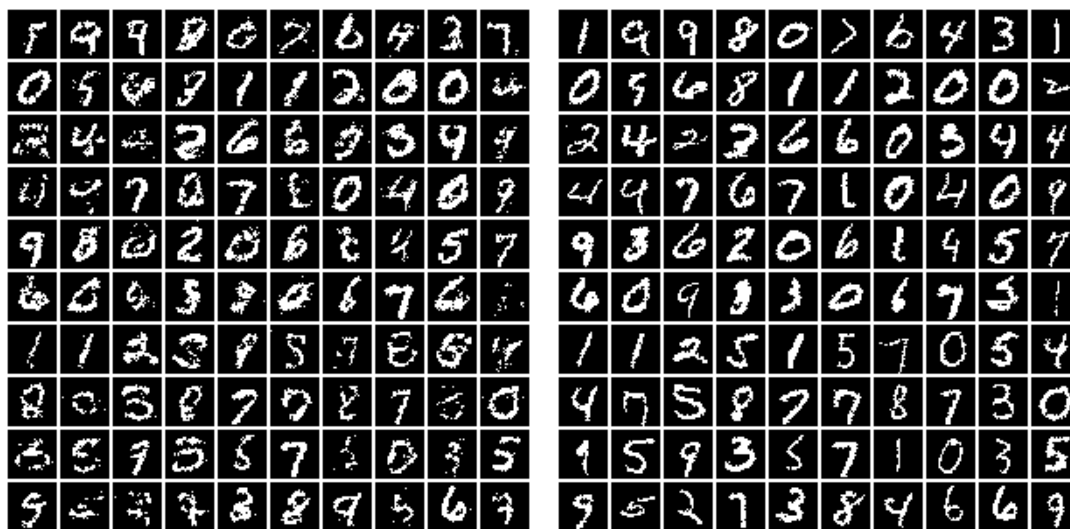
Figure 3. Left: Samples from a 2 hidden layer MADE. Right: Nearest neighbour in binarized MNIST.

Table 6. Negative log-likelihood test results of different models on the binarized MNIST dataset.

Model	$-\log p$	
RBM (500 h, 25 CD steps)	≈ 86.34	
DBM 2hl	≈ 84.62	
DBN 2hl	≈ 84.55	Intractable
DARN n_h=500	≈ 84.71	
DARN n_h=500, adaNoise	≈ 84.13	
MoBernoullis K=10	168.95	
MoBernoullis K=500	137.64	
NADE 1hl (fixed order)	88.33	
EoNADE 1hl (128 orderings)	87.71	
EoNADE 2hl (128 orderings)	85.10	Tractable
MADE 1hl (1 mask)	88.40	
MADE 2hl (1 mask)	89.59	
MADE 1hl (32 masks)	88.04	
MADE 2hl (32 masks)	86.64	

7. Conclusion

We proposed MADE, a simple modification of autoencoders allowing them to be used as distribution estimators. MADE demonstrates that it is possible to get direct, cheap estimates of high-dimensional joint probabilities, from a single pass through an autoencoder. Like standard autoencoders, our extension is easy to vectorize and implement on GPUs. MADE can evaluate high-dimensional probably distributions with better scaling than before, while maintaining state-of-the-art statistical performance.

Acknowledgments

References

Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian J., Bergeron, Arnaud, Bouchard, Nicolas, and Bengio, Yoshua. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Bengio, Yoshua and Bengio, Samy. Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pp. 400–406. MIT Press, 2000.

Bengio, Yoshua, Laufer, Eric, Alain, Guillaume, and Yosinski, Jason. Deep generative stochastic networks trainable by backprop. In *Proceedings of the 31th Annual International Conference on Machine Learning (ICML 2014)*, pp. 226–234. JMLR.org, 2014.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Dinh, Laurent, Krueger, David, and Bengio, Yoshua. NICE: non-linear independent components estimation, 2014. arXiv:1410.8516v2.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. Technical report, EECS Department, University of California, Berkeley, Mar 2010.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pp. 2672–2680. Curran Associates, Inc., 2014.

Gregor, Karol and LeCun, Yann. Learning representations by maximizing compression, 2011. arXiv:1108.1169v1.

Gregor, Karol, Danihelka, Ivo, Mnih, Andriy, Blundell, Charles, and Wierstra, Daan. Deep AutoRegressive Networks. In *Proceedings of the 31th Annual International Conference on Machine Learning (ICML 2014)*, pp. 1242–1250. JMLR.org, 2014.

Kingma, Diederik P. and Welling, Max. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014)*, 2014.

Larochelle, Hugo and Murray, Iain. The neural autoregressive distribution estimator. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, volume 15, pp. 29–37, Ft. Lauderdale, USA, 2011. JMLR W&CP.

Poon, Hoifung and Domingos, Pedro. Sum-product networks: A new deep architecture. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pp. 337–346, 2011.

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning (ICML 2014)*, pp. 1278–1286, 2014.

Salakhutdinov, Ruslan and Murray, Iain. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pp. 872–879. Omnipress, 2008.

Schmah, Tanya, Hinton, Geoffrey E., Zemel, Richard S., Small, Steven L., and Strother, Stephen C. Generative versus discriminative training of RBMs for classification of fMRI images. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pp. 1409–1416, 2009.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Uria, Benigno, Murray, Iain, and Larochelle, Hugo. RNADE: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pp. 2175–2183, 2013.

Uria, Benigno, Murray, Iain, and Larochelle, Hugo. A deep and tractable density estimator. In *Proceedings of the 31th International Conference on Machine Learning, (ICML 2014)*, pp. 467–475, 2014.

Uria, Benigno, Murray, Iain, Renals, Steve, and Valentini-Botinhao, Cassia. Modelling acoustic feature dependencies with artificial neural networks: Trajectory-RNADE. In *Proceedings of the 40th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2015)*. IEEE Signal Processing Society, 2015.

Wan, Li, Zeiler, Matthew D., Zhang, Sixin, LeCun, Yann, and Fergus, Rob. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, (ICML 2013)*, pp. 1058–1066, 2013.

Zeiler, Matthew D. ADADELTA: an adaptive learning rate method, 2012. arXiv:1212.5701v1.